Functions and Like

by Erol Seke

For the course “Introduction to VHDL”

ESKİŞEHİR OSMANGAZI UNIVERSITY

Function

Functions are used to;

1. Improve readability
2. Share/Reuse commonly encountered circuits

Functions are placed in

1. Architecture (declaration section) of the code
2. Entity declaration
3. Package body (of libraries)

```
function Function_Name(<parameter list>) return return_type is
  <declarations>
begin
  sequential_statements;
end Function_Name;
```

Any sequential statements like `if`, `case`, `loop`, but `wait` is excluded.

No signal declaration or component instantiation.

`;` delimited `signal` or `constant` types can be parameters but with no range specifications

`signal A, B: STD_LOGIC_VECTOR`

is ok, but

`constant X: integer ~~range 0 to 10~~`

is not

Examples

```
function Positive_Edge(signal S: STD_LOGIC) return BOOLEAN is
begin
  return (S'event and S='1');
end Positive_Edge;

function conv_integer(signal V: STD_LOGIC_VECTOR)
return integer is
  variable i: integer range 0 to 2**V'length-1;
begin
  if(V(V'high)='1') then i:=1; else i:=0; end if;
  for b in (V'high-1) downto (V'low) loop
    i:=i*2;
    if(V(b)='1') then i:=i+1; end if;
  end loop
  return i;
end conv_integer;
```

```
          …
          signal X,Y: integer range 0 to 255;
          signal U,V: STD_LOGIC_VECTOR (7 downto 0);
          …
```

```
X <= U; -- error (incompatible types)
Y <= conv_integer(V); -- ok
```

Example `EvenParity`

```
entity functionex is Port (
  Din   : STD_LOGIC_VECTOR(7 downto 0);
  Pout : out STD_LOGIC;
  Din2 : STD_LOGIC_VECTOR(15 downto 0);
  Pout2: out STD_LOGIC);
end functionex;

architecture functionex of functionex is
  function EvenParity(A: STD_LOGIC_VECTOR) return STD_LOGIC is
    variable T : STD_LOGIC_VECTOR(A'LENGTH downto 0);
  begin
    T(0)  := '0';
    L1: for i in 0 to A'LENGTH-1 loop
      T(i+1)  := T(i) xnor A(i);
    end loop;
    return T(T'LENGTH-1);
  end EvenParity;

begin

  Pout  <= EvenParity(Din);
  Pout2 <= EvenParity(Din2);

end functionex;
```

repeated generation of logic.
`GENERATE` can not be used here

Homework: Check RTL schematics.
What is the output of
`EvenParity(Din & EvenParity(Din))`

Procedure

```
procedure Procedure_Name(<parameter list>) is
  <declarations>
begin
  sequential_statements;
end Procedure_Name;
```

Similar to function, with some exceptions

Parameters can be `constant`, `signal` or `variable`

Parameter directions can be `in`, `out` or `inout`

`signal` declarations can be made when `procedure` is defined in a `process`

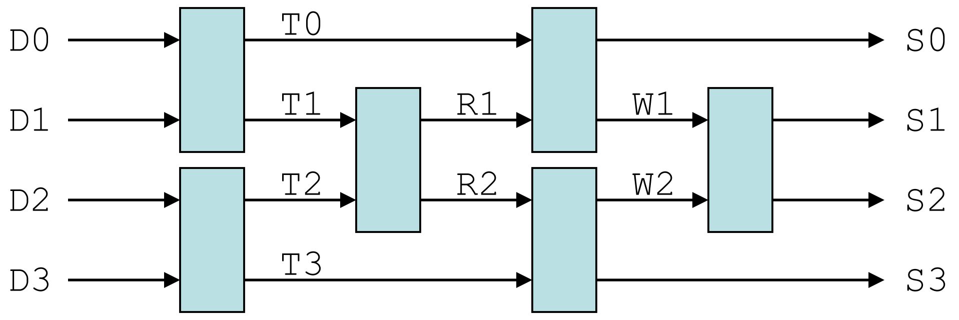
`wait` can not be used, nor any register inferring code (no `'event`)

Procedures can not be used in an expression (unlike functions), but stay with their own.

```
A <= MyFunction(B, C) + YourFunction(X);  -- ok
MyProcedure(A, B, 16); -- ok
X <= YourProcedure(Z); -- not allowed. No return for procedures
```

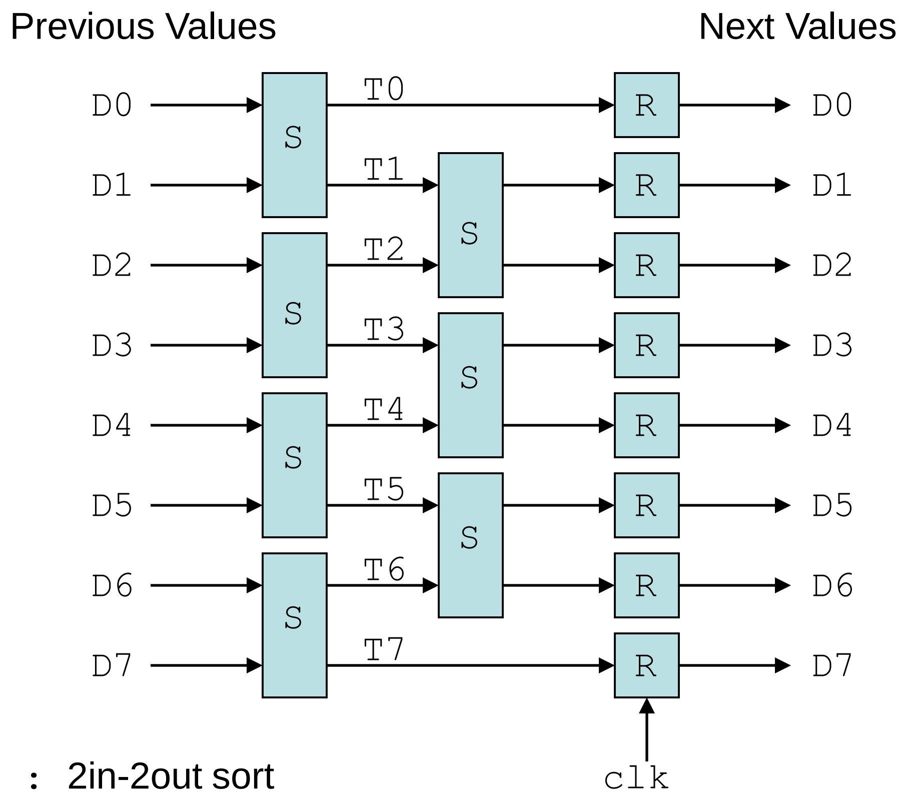
Example

```vhdl
entity procedurex is Port (
  D0, D1, D2, D3 : in  integer range 0 to 255;
  S0, S1, S2, S3 : out integer range 0 to 255 );
end procedurex;

architecture procedurex of procedurex is
  procedure Sort4(signal A0,A1: in integer; signal B0,B1: out integer) is
  begin
    if(A0<A1) then
      B0<=A0; B1<=A1;
    else
      B0<=A1; B1<=A0;
    end if;
  end Sort4;
  signal T0,T1,T2,T3,R1,R2,W1,W2: integer range 0 to 255;
begin

  Sort4(D0,D1,T0,T1);
  Sort4(D2,D3,T2,T3);
  Sort4(T1,T2,R1,R2);
  Sort4(T0,R1,S0,W1);
  Sort4(R2,T3,W2,S3);
  Sort4(W1,W2,S1,S2);

end procedurex;
```



Homework : How about a synchronous (but with less Sort4) design?

Example



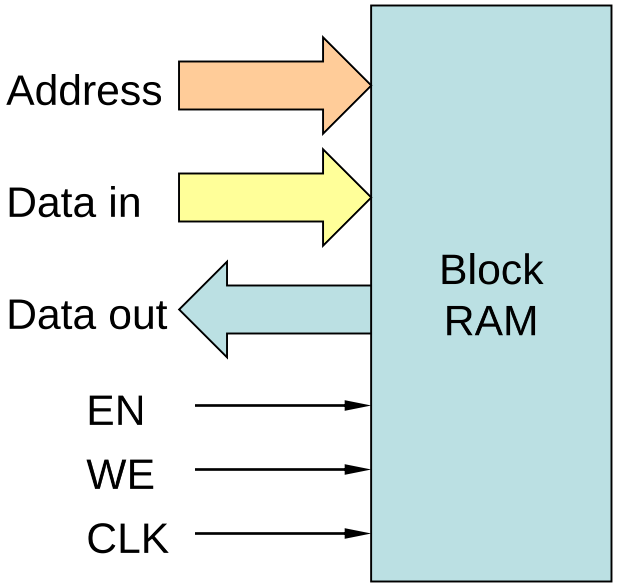
S : 2in-2out sort

R : Register

Homework : Design with VHDL
Determine the number of `clk` cycles required to complete the sort.

Use of BRAM Primitives in FPGAs

There are several RAM blocks in Xilinx FPGAs including Spartan-3E
(and in other vendors' too)



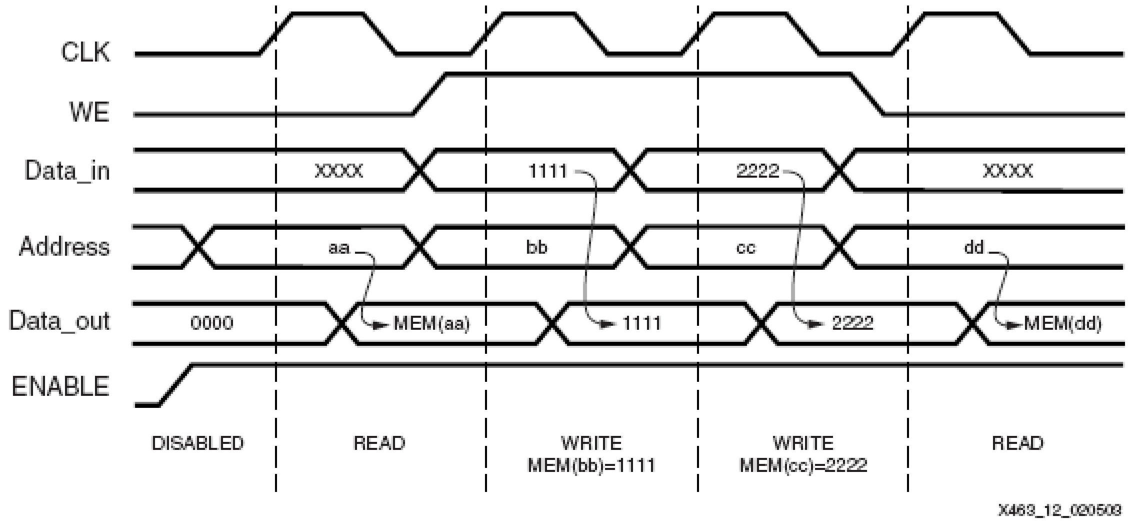
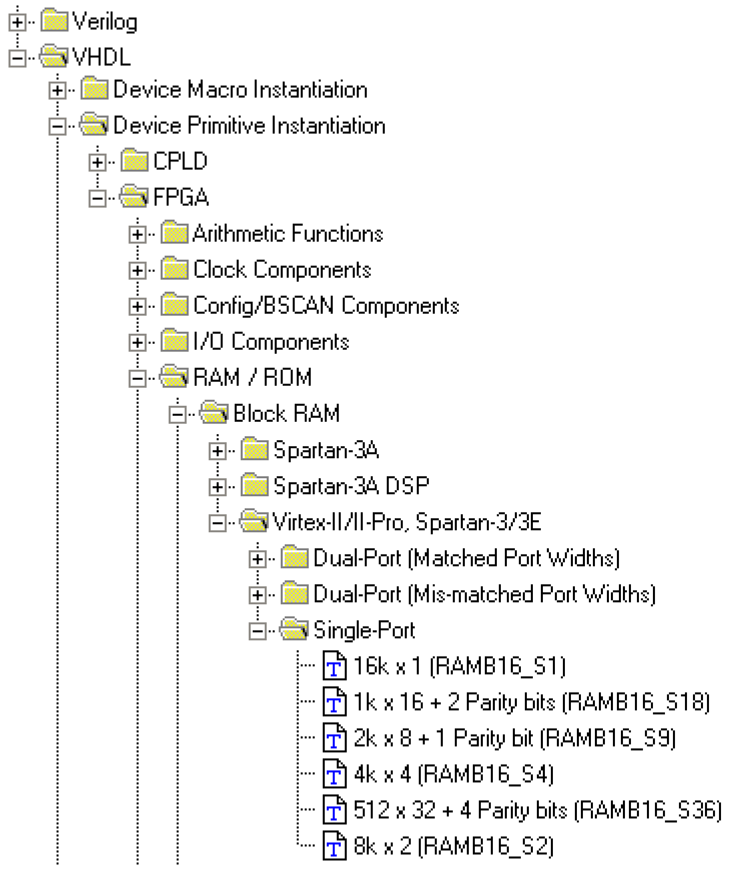


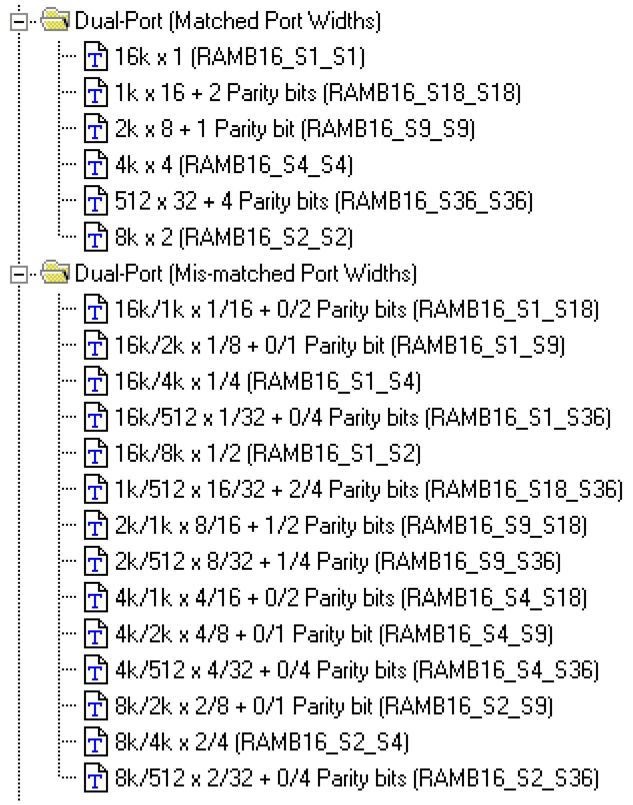
Figure 4-12: **WRITE_FIRST Mode Waveforms**

Available Configurations in Spartan-3E

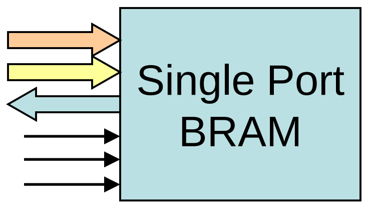
Block RAM instantiation templates can be seen using *Language Templates*

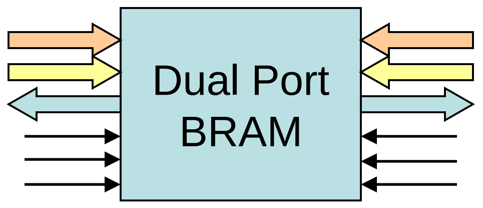


- Verilog
- VHDL
  - Device Macro Instantiation
  - Device Primitive Instantiation
    - CPLD
    - FPGA
      - Arithmetic Functions
      - Clock Components
      - Config/BSCAN Components
      - I/O Components
      - RAM / ROM
        - Block RAM
          - Spartan-3A
          - Spartan-3A DSP
          - Virtex-II/II-Pro, Spartan-3/3E
            - Dual-Port (Matched Port Widths)
            - Dual-Port (Mis-matched Port Widths)
            - Single-Port
              - 16k x 1 (RAMB16_S1)
              - 1k x 16 + 2 Parity bits (RAMB16_S18)
              - 2k x 8 + 1 Parity bit (RAMB16_S9)
              - 4k x 4 (RAMB16_S4)
              - 512 x 32 + 4 Parity bits (RAMB16_S36)
              - 8k x 2 (RAMB16_S2)

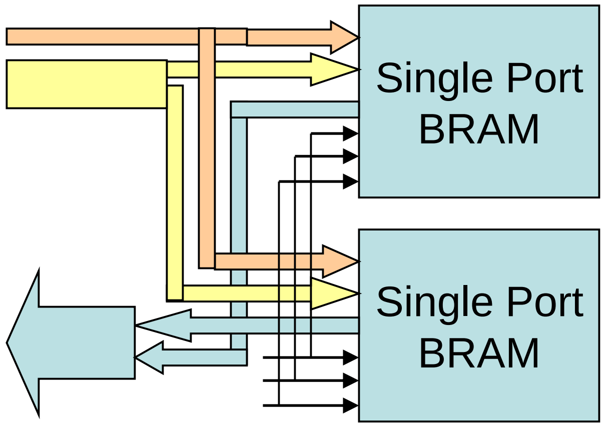


- Dual-Port (Matched Port Widths)
  - 16k x 1 (RAMB16_S1_S1)
  - 1k x 16 + 2 Parity bits (RAMB16_S18_S18)
  - 2k x 8 + 1 Parity bit (RAMB16_S9_S9)
  - 4k x 4 (RAMB16_S4_S4)
  - 512 x 32 + 4 Parity bits (RAMB16_S36_S36)
  - 8k x 2 (RAMB16_S2_S2)
- Dual-Port (Mis-matched Port Widths)
  - 16k/1k x 1/16 + 0/2 Parity bits (RAMB16_S1_S18)
  - 16k/2k x 1/8 + 0/1 Parity bit (RAMB16_S1_S9)
  - 16k/4k x 1/4 (RAMB16_S1_S4)
  - 16k/512 x 1/32 + 0/4 Parity bits (RAMB16_S1_S36)
  - 16k/8k x 1/2 (RAMB16_S1_S2)
  - 1k/512 x 16/32 + 2/4 Parity bits (RAMB16_S18_S36)
  - 2k/1k x 8/16 + 1/2 Parity bits (RAMB16_S9_S18)
  - 2k/512 x 8/32 + 1/4 Parity (RAMB16_S9_S36)
  - 4k/1k x 4/16 + 0/2 Parity bits (RAMB16_S4_S18)
  - 4k/2k x 4/8 + 0/1 Parity bit (RAMB16_S4_S9)
  - 4k/512 x 4/32 + 0/4 Parity bits (RAMB16_S4_S36)
  - 8k/2k x 2/8 + 0/1 Parity bit (RAMB16_S2_S9)
  - 8k/4k x 2/4 (RAMB16_S2_S4)
  - 8k/512 x 2/32 + 0/4 Parity bits (RAMB16_S2_S36)

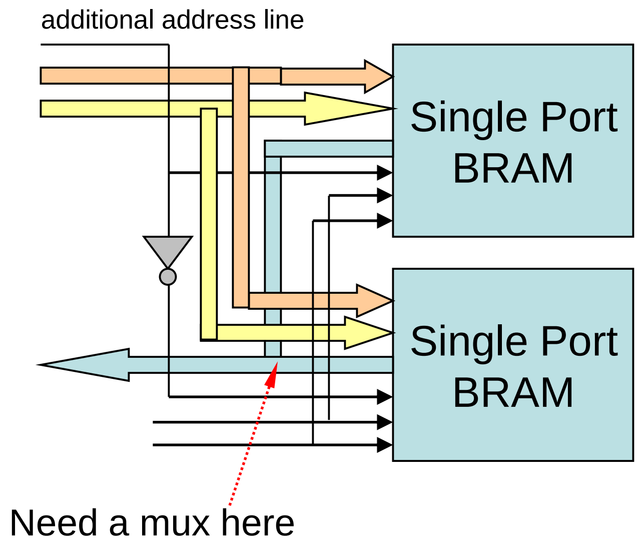




Combining / Extending BRAMs



Data Bus (Horizontal) Extension

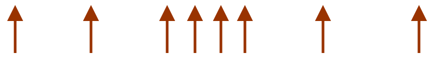


Vertical Extension (more addresses)

Example using Distributed ROMs

We are to compare two binary strings and find the number of matches

```
010001110101101110
111100011011010011
```



Number of matches = 8

This operation corresponds to Correlation (integration) in communications

Actual correlation is performed between incoming stream samples and local data. But here we shall compare two local stream to keep it simple

```
process(clk) is begin
   if(rising_edge(clk)) then
     if(ADR="000000") then
       LEDS <= conv_std_logic_vector(bcor,8);
       if(O1=O2) then bcor <= 1; else bcor <= 0; end if;
     else
       if(O1=O2) then bcor <= bcor + 1; end if;
     end if;

     ADR <= ADR +1;
   end if;
 end process;
```

Declaration of Distributed ROMs

```
  ROM1 : ROM64X1
  generic map (
    INIT => X"0123456789ABCDEF")
  port map (
    O => O1,        -- 1-bit data output
    A0 => ADR(0), -- Address[0] input bit
    A1 => ADR(1), -- Address[1] input bit
    A2 => ADR(2), -- Address[2] input bit
    A3 => ADR(3), -- Address[3] input bit
    A4 => ADR(4), -- Address[4] input bit
    A5 => ADR(5)  -- Address[5] input bit
);
```

```
  ROM2 : ROM64X1
   generic map (
     INIT => X"FEDCBA9876543210")
   port map (
     O => O2,
     A0 => ADR(0),
     A1 => ADR(1),
     A2 => ADR(2),
     A3 => ADR(3),
     A4 => ADR(4),
     A5 => ADR(5)
);
```

Question : What should we see on the LEDs ?

```
function Corr(signal A,B,Cp: in STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
begin
   return (Cp+A*B); --multiply and add function to be used in correlation calculation
end Corr;
```

Defining RAMs with Inference

```
entity DualPortRAM is
    Port ( ADDR_A   : in  STD_LOGIC_VECTOR (6 downto 0);
           DATAIN   : in  STD_LOGIC_VECTOR (7 downto 0);
           WE       : in  STD_LOGIC;
           CLK      : in  STD_LOGIC;
           DATAO_A  : out STD_LOGIC_VECTOR (7 downto 0);
           ADDR_B   : in  STD_LOGIC_VECTOR (6 downto 0);
           DATAO_B  : out STD_LOGIC_VECTOR (7 downto 0));
end DualPortRAM;
architecture Behavioral of DualPortRAM is
  type TDPRAM is array (0 to 127) of std_logic_vector (7 downto 0);
  signal MRAM : TDPRAM;
...
```

```
process(CLK) begin
  if (rising_edge(CLK)) then
    if (WE = '1') then
      MRAM(conv_integer(ADDR_A)) <= DATAIN ;
    end if;
    DATAO_A <= MRAM(conv_integer(ADDR_A));
    DATAO_B <= MRAM(conv_integer(ADDR_B));
  end if;
end process;
```

Synthesizer uses appropriate BRAMS if it can

Dual Port RAM

```vhdl
entity DualPortRAM is Generic (H:integer; W:integer); Port (
 ADDRA  : in  integer range 0 to H-1;
  DINA  : in  STD_LOGIC_VECTOR(W-1 downto 0);
  WEA   : in  STD_LOGIC;
  CLKA  : in  STD_LOGIC;
  DOUTA : out STD_LOGIC_VECTOR(W-1 downto 0);
 ADDRB  : in  integer range 0 to H-1;
  DINB  : in  STD_LOGIC_VECTOR(W-1 downto 0);
  WEB   : in  STD_LOGIC;
  CLKB  : in  STD_LOGIC;
  DOUTB : out STD_LOGIC_VECTOR (W-1 downto 0);
end DualPortRAM;
architecture Behavioral of DualPortRAM is
  type TDPRAM is array (0 to H-1) of STD_LOGIC_VECTOR(W-1 downto 0);
  shared variable MRAM : TDPRAM;
...
```

```vhdl
process(CLKA) begin
  if (rising_edge(CLKA)) then
    if (WEA = '1') then MRAM(ADDRA) <= DINA; end if;
    DOUTA <= MRAM(ADDRA);
  end if;
end process;
process(CLKB) begin
  if (rising_edge(CLKB)) then
    if (WEB = '1') then MRAM(ADDRB) <= DINB; end if;
    DOUTB <= MRAM(ADDRB);
  end if;
end process;
```

Initializing RAM/ROM

1. Manually type in values (or copy-paste from somewhere else)

```
signal MRAM : TDPRAM := (1,3,... etc );
```

2. Use data generation functions

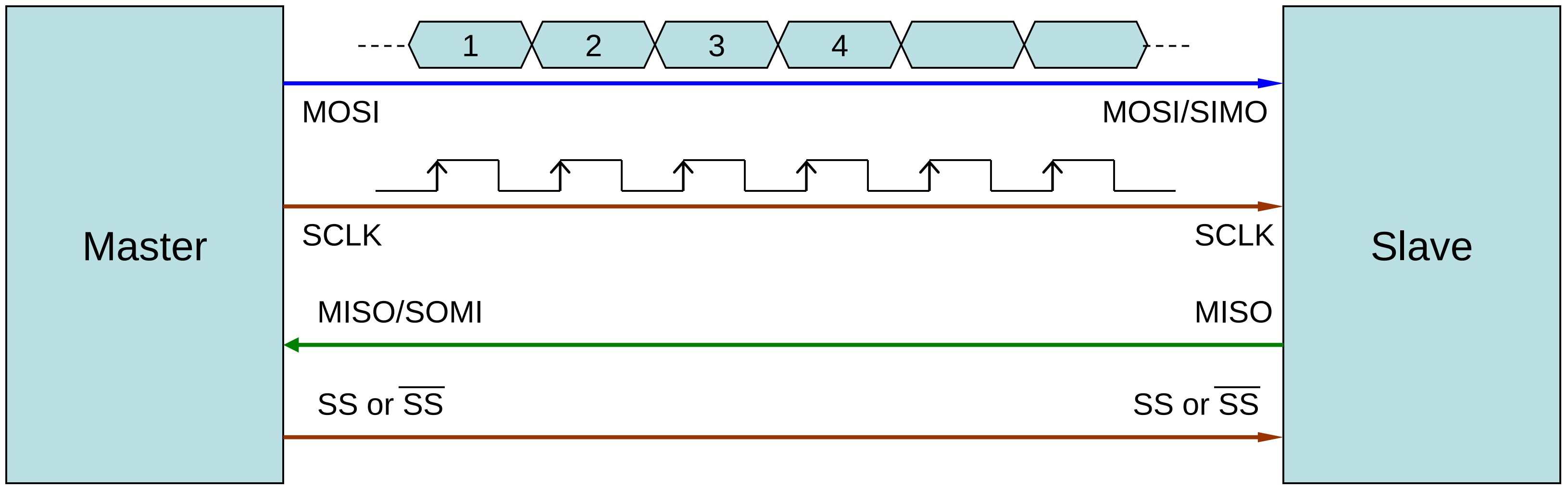
```
use ieee.math_real.all;
...

type Rtype is array (0 to MEMSIZE-1) of integer range -NUMBERMAX to NUMBERMAX;

function initR(N : integer) return Rtype is
  variable rv : Rtype;
begin
  for i in 0 to N-1 loop
    rv(i) := integer(round(real(NUMBERMAX)*cos(MATH_2_PI*real(i)/real(N))));
  end loop;
  return rv;
end function;

signal MRAM : Rtype := := initR(MEMSIZE);
```

3. Read from a file

```
homework
```

Serial Peripheral Interface (SPI)

Transmit: Master puts the bit to be transmitted and raises the clock (SCLK)
Slave latches the serial input on the rising edge of the incoming clock.



Receive: Master raises the SCLK and reads in MISO on the falling edge of SCLK.

SPI can not be used in long distance (a couple meters) communications, because of the timing problems. The length difference between clock and data lines must be much smaller than the clock wavelength.
SPI is used for board to board or chip to chip data transfers

A Test Case

```
entity SPI is Port (
  clk    : in  STD_LOGIC;  --we need a clock for action
  ------------- transmitter part
  SWS    : in  STD_LOGIC_VECTOR(3 downto 0); -- parallel data in
  clkout : out STD_LOGIC;                    -- SPI clock
  Sout   : out STD_LOGIC;                    -- serial data out
  ------------- receiver part
  clkin  : in  STD_LOGIC;                    -- SPI clock in
  Sin    : in  STD_LOGIC;                    -- serial data in
  LEDS   : out STD_LOGIC_VECTOR(3 downto 0));-- parallel data out
end SPI;
```

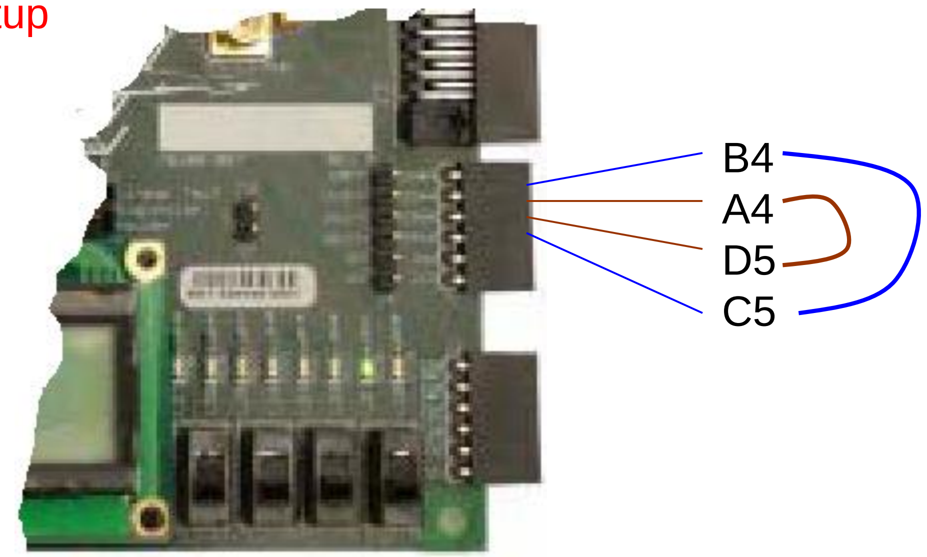
```
 signal cntr,reccntr : integer range 0 to 3;
 signal recsig : STD_LOGIC_VECTOR(3 downto 0);
```

```
clkout <= clk;
TRNS: process(clk) is begin
  if(falling_edge(clk)) then
    Sout <= SWS(cntr);
    cntr <= cntr+1;
  end if;
end process;
```

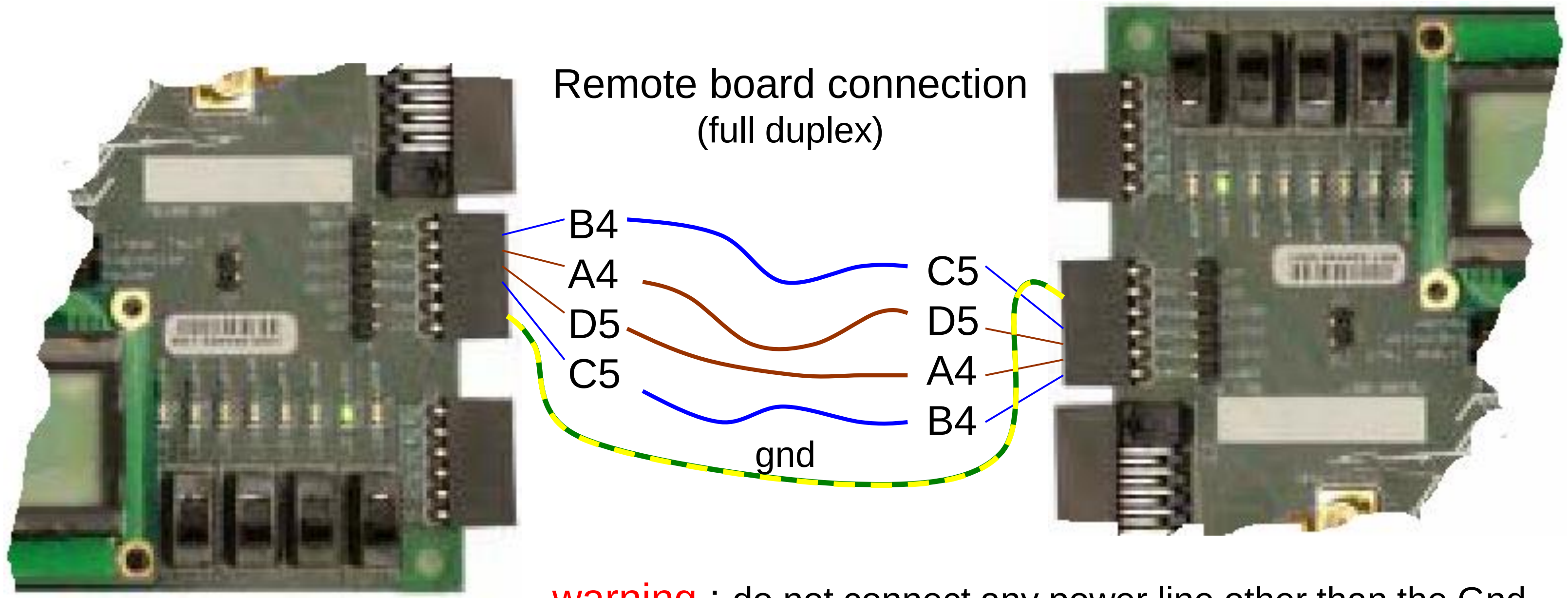
```
RECV: process(clkin) is begin
  if(rising_edge(clkin)) then
    recsig(reccntr) <= Sin;
    if(reccntr=3) then
      LEDS <= recsig;
    end if;
    reccntr <= reccntr +1;
  end if;
end process;
```

Setup

Single board loopback test connection



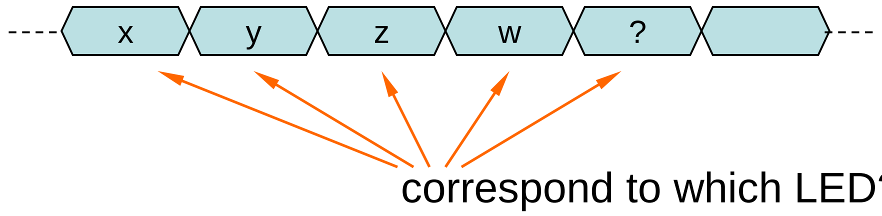
Remote board connection (full duplex)



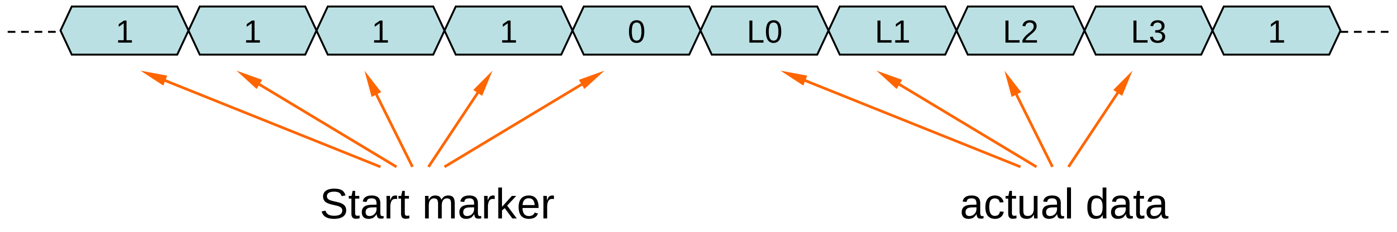
warning : do not connect any power line other than the Gnd

What do we see in the tests?

We do not know which switch actually corresponds to which LED.
Because there is no reference marker in our simple protocol.



Simple remedy : insert a start marker before the actual data



You may choose your own start marker pattern and length in order for the easier detection of the marker.

```
TRNS: process(clk) is begin
  if(falling_edge(clk)) then
    if(cntr<4) then
      Sout <= '1';
    elsif(cntr=4) then
      Sout <= '0';
    else
      Sout <= SWS(cntr-5);
    end if;
    if(cntr=8) then
      cntr <= 0;
    else
      cntr <= cntr+1;
   end if;
  end if;
end process;
```

```
function bit_reverse (A : in std_logic_vector)
return std_logic_vector is
  constant L : natural := A'length;
  variable aa, yy : std_logic_vector(L-1 downto 0);
begin
  aa := A;
  for i in aa'range loop
    yy(i) := aa(L - 1 - i);
  end loop;
  return yy;
end bit_reverse;

function shiftleft (A : in std_logic_vector)
return std_logic_vector is
  variable B: std_logic_vector(A'HIGH downto 0);
begin
  for i in 0 to A'HIGH-1 loop
    B(i+1) := A(i);
  end loop;
  return B;
end shiftleft;
```

```
RECV: process(clkin) is begin
  if(rising_edge(clkin)) then
    recsig <= shiftleft(recsig);
    recsig(0) <= Sin;
    if(recsig(8 downto 4)="11110") then
      LEDS <= bit_reverse(recsig(3 downto 0));
    end if;
  end if;
end process;
```

We have a 9 bit shift register. Serial data is shifted in from right. When the leftmost 5 bits equals to the pattern then the 4 rightmost bits are output to the LEDs.

Q: What is the potential problem here?

END